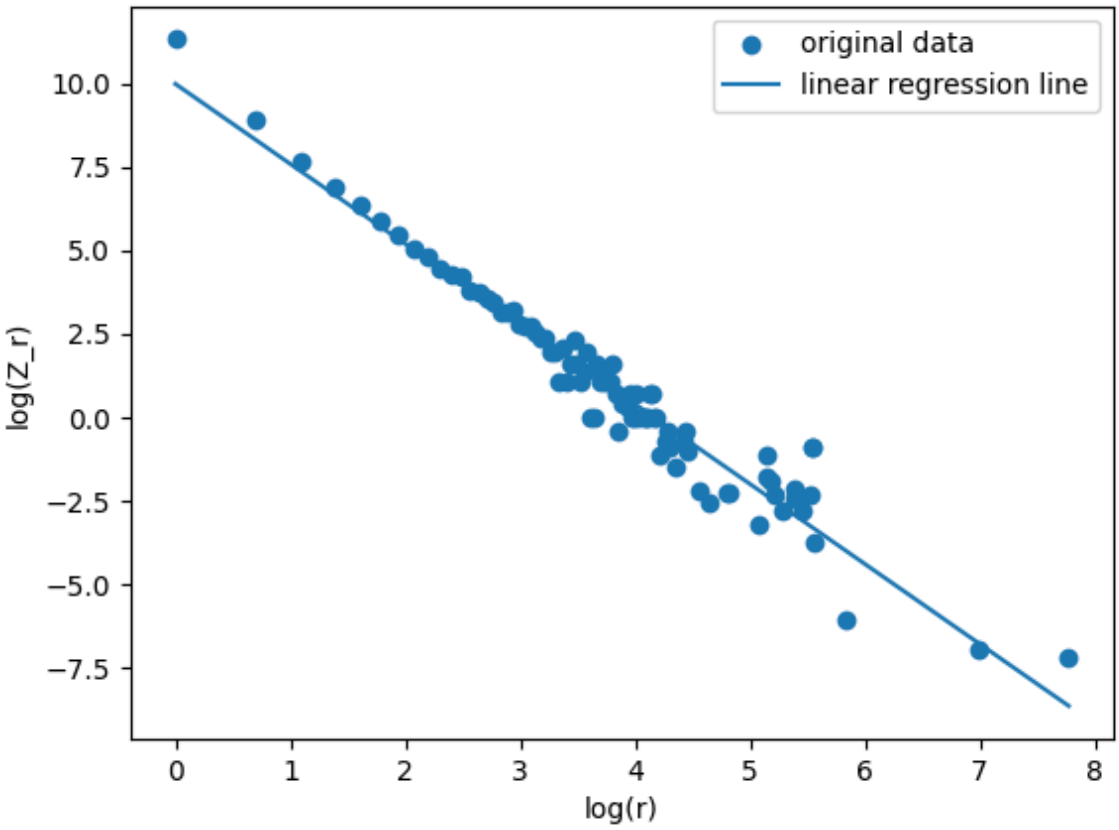
iNLP

Assignment-1 Report

Akshit Sharma
2021101029

Good Turing Smoothing



Regression Line fitted to Frequency of Frequency Data

Perplexities obtained for different model:

- **LM1 (Tokenisation + 3-gram LM + Good-Turing Smoothing for 'Pride and Prejudice' corpus)**

  Test Set (1000 random sentences) : Average Perplexity=96756.25

  Train Set (remaining sentences) : Average Perplexity=21882002.2

- **LM2 (Tokenisation + 3-gram LM +Linear Interpolation for 'Pride and Prejudice' corpus)**

  Test Set (1000 random sentences) : Average Perplexity=357.82

  Train Set (remaining sentences) : Average Perplexity=44.553

- **LM3 (Tokenisation + 3-gram LM + Good-Turing Smoothing for 'Ulysses' corpus)**

Test Set (1000 random sentences) : Average Perplexity=149227.92

Train Set (remaining sentences) : Average Perplexity=48551795.7

- **LM4 (Tokenisation + 3-gram LM +Linear Interpolation for 'Ulysses' corpus)**

Test Set (1000 random sentences) : Average Perplexity=325.16

Train Set (remaining sentences) : Average Perplexity=37.897

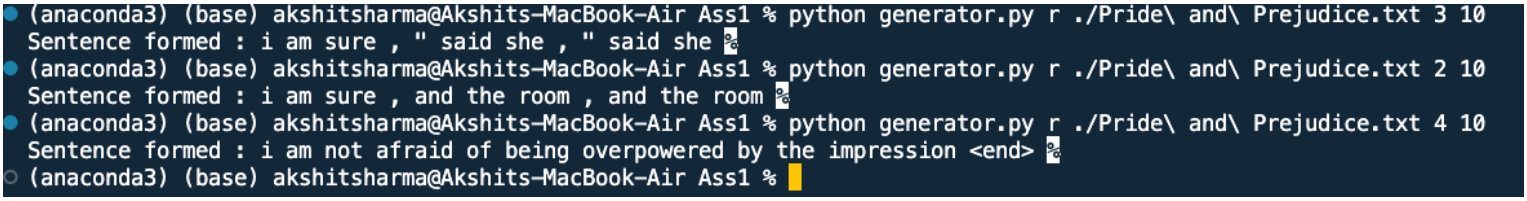
Generation

```
(anaconda3) (base) akshitsharma@Akshits-MacBook-Air Ass1 % python generator.py n ./Pride\ and\ Prejudice.txt 5
input sentence: who is
output:
to 0.18181818181818182
good 0.09090909090909091
likely 0.09090909090909091
more 0.09090909090909091
supposed 0.09090909090909091
(anaconda3) (base) akshitsharma@Akshits-MacBook-Air Ass1 % python generator.py n ./Pride\ and\ Prejudice.txt 5 2
input sentence: who is
output:
a 0.08618504435994931
not 0.08365019011406843
the 0.043092522179974654
very 0.036755386565272496
to 0.029150823827629912
(anaconda3) (base) akshitsharma@Akshits-MacBook-Air Ass1 % python generator.py n ./Pride\ and\ Prejudice.txt 5 4
input sentence: who is
output:
to 1.0
```

```
(anaconda3) (base) akshitsharma@Akshits-MacBook-Air Ass1 % python generator.py n ./Pride\ and\ Prejudice.txt 5 4
input sentence: there is a
output:
mixture 0.125
stubbornness 0.125
fine 0.125
strong 0.125
real 0.125
(anaconda3) (base) akshitsharma@Akshits-MacBook-Air Ass1 % python generator.py n ./Pride\ and\ Prejudice.txt 5
input sentence: there is a
output:
most 0.11764705882352941
very 0.11764705882352941
great 0.04411764705882353
question 0.029411764705882353
handsome 0.029411764705882353
(anaconda3) (base) akshitsharma@Akshits-MacBook-Air Ass1 % python generator.py n ./Pride\ and\ Prejudice.txt 2
input sentence: there is a
output:
most 0.11764705882352941
very 0.11764705882352941
```

From multiple runs on the N-gram model generation, we see that for higher values of N, we see that it gives a more fluent prediction. As you can see in the above example, when we give input as 'there is a' to a 4-gram model compared to 3-gram and 2-gram ones, we see that 'there is a fine' , 'there is a strong', etc. make more sense in terms of fluency as compared to 'there is a most' in 3-gram model or 'there is a very' for the 2 -gram model.

Generating Sentences from start



As n for the N-gram model increases, the generated sentence becomes more fluent/meaningful as the next word becomes more and more constrained.